Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра информационно-аналитических систем

Кокорев Андрей Дмитриевич

Разработка PostgreSQL сервиса с резервным копированием и восстановлением в облачной платформе Cloud Foundry: локальное хранилище и модуль резервного копирования

Бакалаврская работа

Научный руководитель:
к. ф.-м. н., доцент Михайлова Е. Г.

Рецензент:
Директор по технологиям и инновациям, EMC Павел Егоров

Санкт-Петербург
2016

Saint Petersburg University
Mathematics And Mechanics faculty

Chair of Analytical Information Systems

Andrey Kokorev

Development of PostgreSQL service with backup and recovery for Cloud Foundry: local storage and backup module

Bachelor's Thesis

Scientific supervisor:
associate professor Elena Mikhailova

Reviewer:
Engineering Director/CTO at EMC Pavel Egorov

Saint Petersburg
2016

Оглавление

Введение

Третья платформа

Соверменное программное обеспечение с течением времени перемещается в облака. Этот процесс начался с различных Web-приложений, перемещающих различные задачи с компьютера пользователя на удаленный сервер. С течением времени, все больше задач стало удобнее и дешевле выполнять на удаленной инфраструктуре. Причем детали самой инфраструктуры совершенно не заботят конечного пользователя, его интересует лишь результат. В связи с этим появились такие понятия как IaaS (Infrastructure-as-a-Service), SaaS (Software-as-a-Service), PaaS (Platform-as-a-Service). Такая модель переносит вычисления в некоторое "облако", что приводит к так называемой Третьей платформе. Это понятие объединяет современные технологии: облачные технологий и сервисы, аналитику больших данных, доступ к корпоративным инфраструктурам при помощи мобильных устройств, а также социальные сети. Одной из таких современных облачных платформ является Cloud Foundry[5].

Cloud Foundry

Cloud Foundry — современная облачная платформа (PaaS) с открытым исходным кодом. Основная задача этой платформы — ускорение и удешевления процесса разработки и использования приложений. Это включает в себя процессы компиляции, тестирования, развертки и масштабирования приложений. В разработке Cloud Foundry учавствуют несколько крупных IT-корпораций и позиционируют платформу как "продукт от лидеров индустрии для лидеров индустрии" [5].

На основе открытой Cloud Foundry существуют различные коммерческие платформы, такие как IBM Bluemix[10], Pivotal Cloud Foundry[11], SAP HANA Cloud Platform[13] и другие, которые расширяют и увеличивают возможности открытой платформы. Такой подход позволяет компаниям с одной стороны предлагать уникальные возможности сво-

ей платформы, а с другой — иметь достаточно унифицированные внутренние интерфейсы и структуры, упрощающие обеспечивание совместимость пользовательских приложений с разными платформами. Это утверждение в основном верно и для сервисных приложений, обеспечивающих работу платформы.

Подводя итог, разработка сервисных приложений для открытой платформы Cloud Foundry (т.е. разработка для платформы, лежащей в основе остальных из рассмотренных выше) обеспечивает возможность переноса на другие платформы с минимальными затратами. В связи с предыдущим утверждением, в этой работе решена задача предоставления сервиса PostgreSQL пользовательским приложениям, развернутым в Cloud Foundry. Предоставляемый сервис включает возможности бэкапа и восстановления в различные хранилища данных.

1. Постановка задачи

1.1. Сервис-брокеры

Облачная платформа Cloud Foundry позволяет приложениям использовать различные сервисы. В качестве сервисов могут выступать различные СУБД, системы мониторинга и тестирования, системы развертки ПО и т.п. Чтобы приложение могло использовать какой-либо сервис, необходим так называемый сервис-брокер[6]. Это программный продукт, осуществляющий связь сервиса, приложения и платформы.

Сервис-брокер осуществляет следующие задачи в рамках платформы:

- оповещение платформы и разработчика приложений о возможностях предоставляемого сервиса;
- оповещение сервиса о необходимости выделения ресурсов приложению;
- ассоциирование конкретного приложения с выделенными ресурсами сервиса;

1.2. Задача

Был проведен анализ существующих решений. На момент написания работы существует всего несколько сервис-брокеров предоставляющих PostgreSQL, например ElephantSQL[9] или открытый PostgreSQL брокер от сообщества[4].

Первый из двух указанных брокеров предоставляет полный комплект возможностей по управлению СУБД, однако позиционируется как SaaS-проект. Это означает, что недостижим полный контроль над нижележащей инфраструктурой, который может требоваться в некоторых случаях. Также внешний сервис в некоторых случаях может не удовлетворять установленным требованиям безопасности и надежности.

Второй вариант — открытый брокер — позволяет использовать собственную инфраструктуру. Открытый исходный код позволяет самостоятельно осуществить запуск на контролиуремой инфраструктуре, а также внести любые правки и совершенствования под свою задачу. Однако этот брокер не обладает важнейшими возможностями бэкапа и восстановления данных.

Таким образом поставлена следующая задача: разработка сервис-брокера, предоставляющего сервис PostgreSQL с возможностями бэкапа и восстановления данных администратором приложения.

2. Общая архитектура

После анализа поставленной задачи, требований платформы и существующих решений была разработана архитектура сервис-брокера. Получена архитектура модульного типа, с компонентами, предназначенными для решения различных задач.

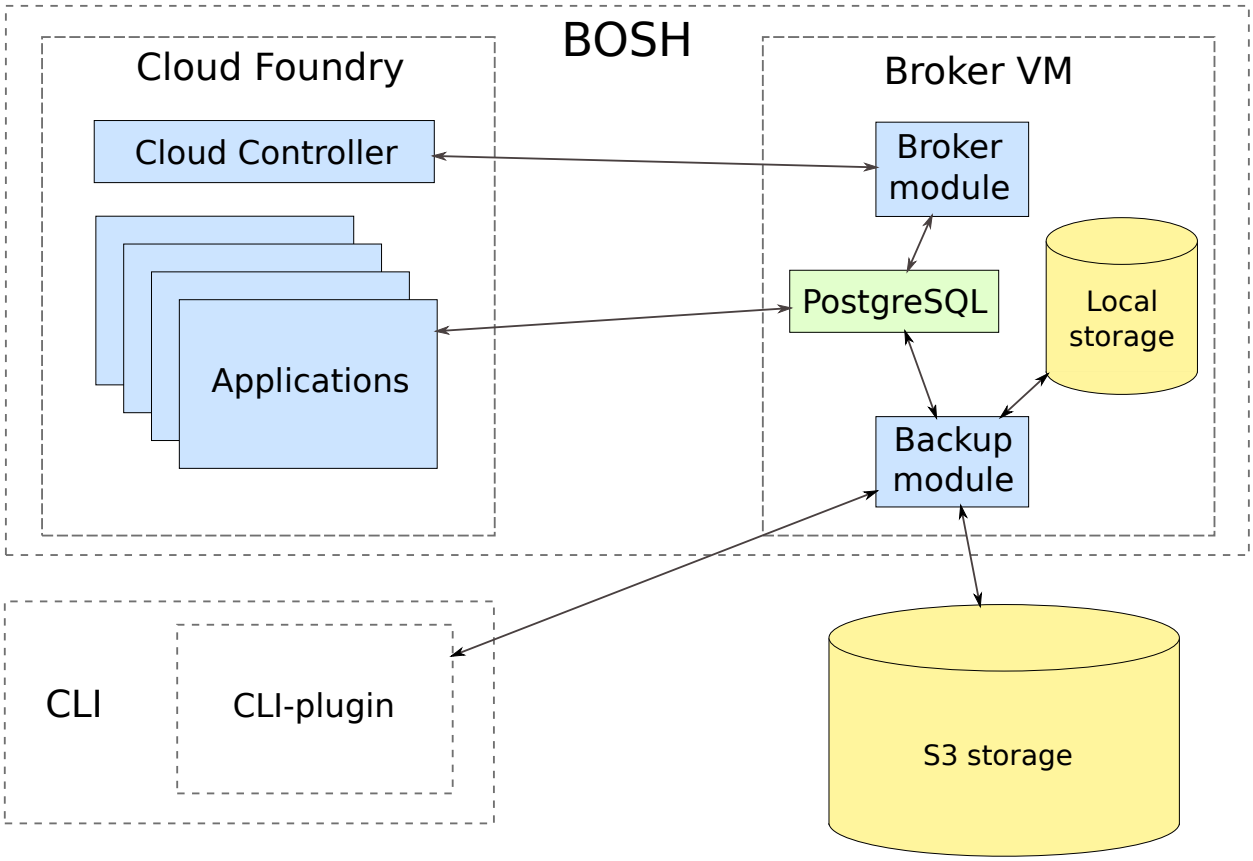

Рис. 1: Общая архитектура

Сервис-брокер состоит из трех основных компонент (Рис. 1):

1. Виртуальная машина брокера, включающая несколько модулей:
   - Брокер (Раздел 3: Брокер);
   - Модуль бэкапа (Раздел 4: Backup-модуль);
   - Локальное хранилище (Раздел 5.2: Локальное хранилище);
   - PostgreSQL;
2. Плагин для командного интерфейса (Раздел 6: Плагин);

3. Внешнее хранилище (Раздел 5.1: Удаленное хранилище);

Эти компоненты независимы друг от друга и вместе составляют полноценный программный продукт.

2.1. Использованные технологии

Основная часть кодовой базы проекта написана на языке Java версии 7. Выбор именно этого языка обусловлен наличием достаточного опыта разработки всех участников проекта на этом языке, существованием удобных инструментов, фреймворков и большого сообщества.

2.1.1. Spring

Для ускорения и упрощения разработки был использован фреймворк Spring[14]. Этот фреймворк позволяет разрабатывать web-сервисы с использованием модели MVC. Большая часть, не зависящая от конкретного приложения уже реализована, что позволяет заниматься разработкой только бизнес-логики.

Так как сервис-брокер предназанчен для запуска на виртуальной машине BOSH, то он требует специальной конфигурации. Однако для разработчика удобнее, если он может запустить брокер на своей рабочей машине. Понятно, что конфигурация брокера в этом случае будет отличаться. Для того, чтобы избежать сложности с постоянным реконфигурированием, а так же различные ошибки связанные с порчей конфигурации были использованы так называемые профили. Профили позволяют иметь различные конфигурации и загружать нужную в зависимости от окружения. Таким образом, был создан профиль *production*, используемый при запуске в виртуальной машине BOSH, а также профили разработчиков для использования на локальных машинах.

2.1.2. Maven

Для автоматизации сборки и управления зависимостями проекта был использован фреймворк Apache Maven[2]. Это один из самых популярных фреймворков в мире Java, используемый повсеместно.

3. Брокер

Брокер — это специализированный веб-сервис. Для его использования платформой Cloud Foundry он должен реализовывать стандартный API, описанный в документации[7]. Этот API описывает протокол взаимодействия платформы и брокера в различных ситуациях. Брокер теоретически может использоваться как самостоятельный продукт, однако в этой работе предполагается его использование на базе платформы. Для того, чтобы это было возможно, разработана специальная конфигурация для использования в менеджере инфраструктуры BOSH[3]. Эта конфигурация описывает виртуальную машину и различные скрипты для запуска и поддерживания работоспособности брокера.

3.1. Внутреннее устройство

Анализ требований CF к брокеру показал, что рационально выделить три задачи: управление экземплярами сервиса, управление сопоставлением приложений и управление конфигурацией и метаинформацией. Для обработки запросов были созданы три соответствующих контроллера (Рис. 2).

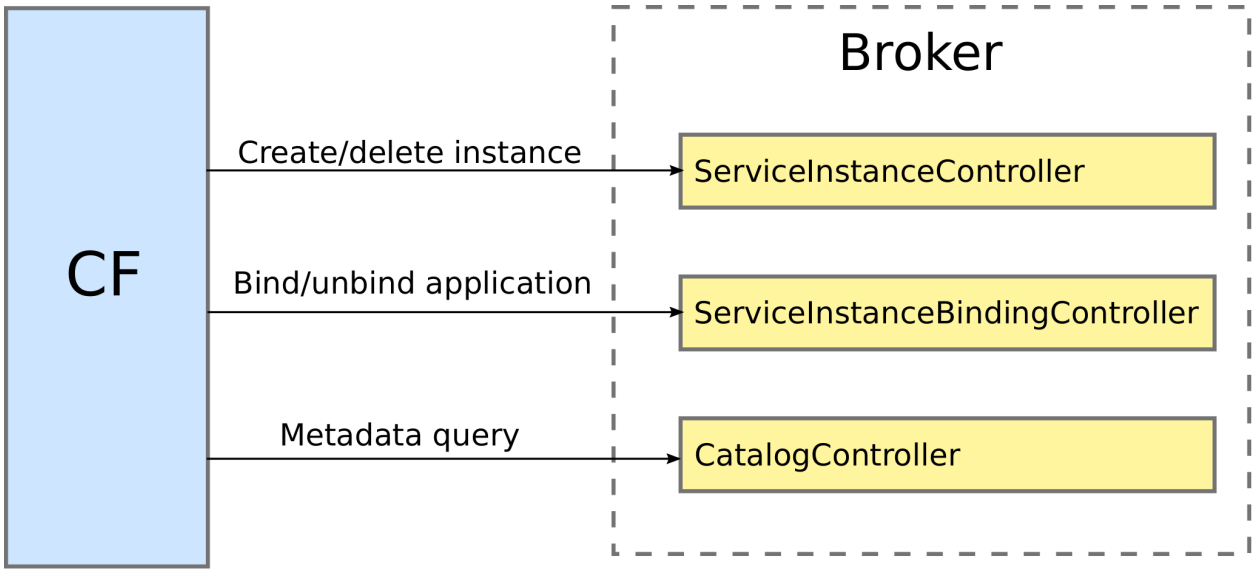

Рис. 2: Контроллеры брокера

Всего API сервис-брокеров в CF описывает 5 видов зарпосов:

1. Создание логического экземпляра сервиса
   ServiceInstanceController

2. Ассоциирование экземпляра сервиса с конкретным приложением
   ServiceInstanceBindingController

3. Удаление ассоциации: экземпляр сервиса $\longleftrightarrow$ приложение
   ServiceInstanceBindingController

4. Удаление экзмепляра сервиса
   ServiceInstanceController

5. Запрос описания предоставляемого сервиса
   CatalogController

Контроллеры производят различные проверки возможности выполнения операций, такие как проверки прав доступа и проверки корректности операции. Затем происходит обращение к PostgreSQL, где производятся нужные действия. Каждый экземпляр предосталяемого сервиса в терминах платформы соответствует базе данных в PostgreSQL. Каждому приложению соответствует пользователь, имеющий права доступа только к конкретной базе данных, привязанной к экземпляру сервиса ассоциированного с этим приложением. Таким образом контроллеры осуществляют запросы создания и удаления баз данных, а так же создания пользователей с нужными правами и их удаление.

4. Backup-модуль

Обработкой запросов бэкапа и восстановления данных занимается backup-модуль брокера. PostgreSQL сам по себе имеет предусмотренную возможность создавать бэкапы и восстанавливаться из них при помощи специальных утилит *pg_ dump* и *pg_ restore*. Использование этих утилит позволит сохранить все гарантии и ограничения, которые они накладывают. Это означает, что администратору приложений не нужно изучать никаких новых правил, скорость работы, консистентность бэкапа и возможность использования этой базы данных на чтение и запись сохраняются[12].

Так как обе утилиты — отдельные программные продукты, то нужно обеспечить их запуск и выполнение, а также сохранение полученных результатов в Java-коде модуля. Таким образом, модуль бэкапа — это некторое промежуточное звено между администратором приложений и утилитами *pg_ dump/pg_ restore*.

Модуль бэкапа, по аналогии с брокером, реализован в виде Web-сервиса. Это позволяет с одной стороны обеспечить API, который будет простым для использования, а с другой стороны общую гомогенность системы (брокер также является web-сервисом).

4.1. Backup API

Для использования модуля бэкапа был разработан специальный API. Через этот API можно запросить создание бэкапа, восстановление данных из существующего бэкапа, указать хранилище, с которым нужно работать, загрузить бэкап в хранилище, выгрузить бэкап из хранилища, а так же посмотреть полный список всех доступных бэкапов в хранилищах.

1. `/backup_instance/instance_id?`
   `storage=storage_type`
   Создать бэкап в указанном хранилище и получить имя созданного бэкапа.

2. `/restore_instance/instance_id/backup_name?storage=storage_type`
   Восстановить данные из указанного файла бэкапа.

3. `/upload_backup/instance_id/backup_name?storage=storage_type`
   Загрузить файл бэкапа в нужное хранилище.

4. `/download_backup/instance_id/backup_name?storage=storage_type`
   Скачать указанный файл бэкапа из хранилища.

5. `/backups/instance_id`
   Получить список доступных бэкапов.

В вышеперечисленных запросах встречаются аргументы:

1. `instance_id`
   Уникальный идентификатор экземпляра сервиса, предоставляется cloud controller'ом платформы CF.

2. `backup_name`
   Имя файла бэкапа, сохраненного в одном из хранилищ.

3. `storage_type`
   Тип хранилища, опциональный аргумент. Может быть одним из: `local`, `s3`.
   Значение по умолчанию — `local`.

5. Хранилища

Хранение полученных бэкапов — не менее важная задача, чем их успешное создание и восстановление. Для хранения данных брокер использует так называемые хранилища. Это программные модули, которые реализуют специальный API хранилищ (Рис. 3). Этот API описывает способы взаимодействия брокера и хранилища:

1. создание раздела
2. удаление раздела
3. добавление в хранилище в указанный раздел
4. получение из хранилища из указанного раздела
5. получение содержимого раздела

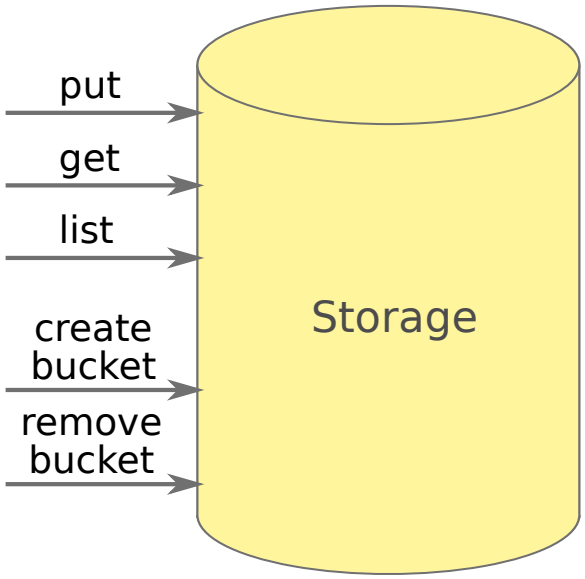

Рис. 3: Интерфейс хранилищ

Указанных выше способов достаточно для полноценной совместной работы хранилища и брокера. Брокер поддерживает прозрачное добавление новых хранилищ, если в этом возникает потребность. Новому хранилищу достаточно реализовать описанный выше API и добавить

свое описание в конфигурацию. Для использования различных хранилищ требуется специальный менеджер хранилищ, который выбирает запрошенное хранилище из доступных и возвращает его для дальнейшего использования.

Опишем действия, которые происходят, когда модуль бэкапа получает запрос на создание новой резервной копии в хранилище `storage` для экзмепляра сервиса `instance`:

1. модуль бэкапа запрашивает StorageManager о хранилище `storage`
2. если хранилище `storage` отсутствует — возвращает ошибку
3. модуль бэкапа начинает создавать резервную копию
4. обращение к `storage`: запрос на добавление
5. если в `storage` отсутствует раздел для `instance`, то раздел создается
6. `storage` создает новый файл и возвращает его
7. модуль бэкапа записывает данные в указанный файл

В этой работе предлагается два различных способа хранения, предназначенные для разных целей. Этим способам соответствуют два хранилища: локальное и удаленное с поддержкой S3-API.

5.1. Удаленное хранилище

Удаленное хранилище состоит из двух компонент: внешнего удаленного хранилиае и промежуточного интерфейса. Брокер поддерживает любое внешнее удаленное хранилище, единственное ограничение — поддержка S3-API[1]. Для преобразования запросов брокера к хранилищу в S3-API запросы используется промежуточный интерфейс.

Это хранилище предназначено для долговременного и надежного хранения данных. В работе не стояло задачи разработки надежного

хранилища, поэтому было решено реализовать поддержку популярного S3-API, как раз предназначенного для коммуницирования с такими хранилищами. Такой подход позволяет с одной стороны предоставить пользователю достаточно широкий выбор, а с другой — упростить задачу и не усложнять программный продукт в целом.

5.2. Локальное хранилище

Локальное хранилище реализует хранение данных на основе локальной файловой системы виртуальной машины брокера. Используется специальная директория, заданная в файле конфигурации брокера. В этой директории производится хранение бэкапов всех существующих экземпляров сервиса. В результате работы получается простая иерархия директорий (Рис. 4).

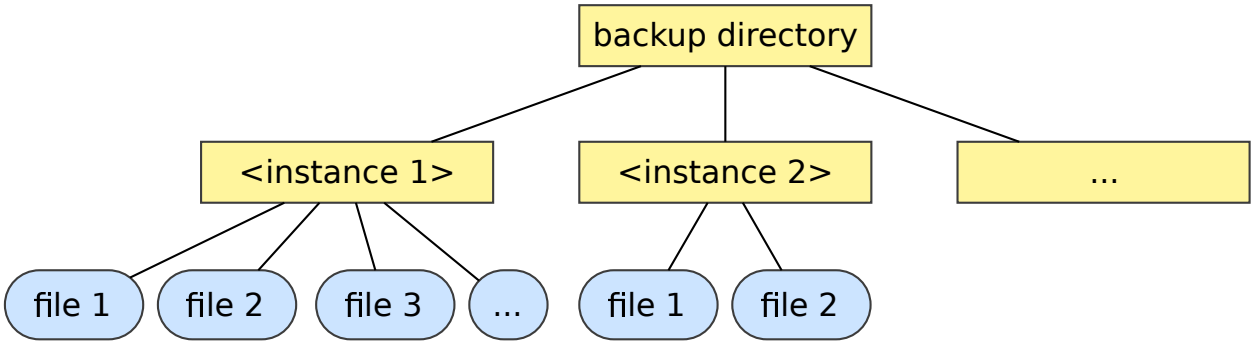

Рис. 4: Файловая система локального хранилища

Это хранилище предназначено для временного хранения бэкапов в случае недоступности внешнего хранилища. Также одним из применений может быть следующее: в ситуации, когда нужно сделанный бэкап выгрузить и куда-либо перенести, нерационально выполнять загрузку в удаленное хранилище, а потом выгрузку оттуда, т.к. бэкап может иметь довольно существенный размер, а пропускная способность канала связи ограничена. Заметно быстрее будет сохранить бэкап в локальное хранилище, т.к. в этом случае не требуется передача данных по сети. Аналогичная ситуация и с обратной задачей: восстановление из бэкапа, которого нет в хранилищах.

6. Плагин

Платформа Cloud Foundry имеет специальный командный интерфейс для ее управления — Cloud Foundry Command Line Interface[8]. Этот интерфейс предоставляет пользователю позможности управления платформой, сервисами и приложениями. Каждый пользователь платформы имеет определенный уровень прав, который обеспечивает доступ к тем или иным компонентам платформы. Для упрощения и автоматизации выполнения некоторых задач в командном интерфейсе предусмотрена возможность выполнения так называемых плагинов. Плагин имеет те же возможности, что и пользователь, т.е. те же права и тот же интерфейс управления платформой, только адаптированный для машинного использования.

Один из основных способов использования плагинов для CF CLI — это расширение возможностей для управления различными сервисами. В этой работе было решено разработать плагин, позволяющий управлять процессами бэкапа, добавив соответствующие команды в командный интерфейс.

Плагин предоставляет соответствующую команду для каждой команды Backup API (Раздел 4.1: Backup API):

1. `cf backup-postgres [--useS3] serviceName`
2. `cf backup-postgres [--useS3] -restore filename serviceName`
3. `cf backup-postgres [--useS3] -download filename serviceName`
4. `cf backup-postgres [--useS3] -upload filename serviceName`
5. `cf backup-postgres -list serviceName`

Заключение

В этой работе была поставлена и решена задача разработки специального программного продукта — сервис-брокера — для предоставления сервиса PostgreSQL приложениям, запускаемым в открытой облачной платформе Cloud Foundry. Проведено исследование современных облачных технологий, в частности облачных платформ, предоставляемых ими сервисов и существующих PostgreSQL сервис-брокеров. В постановку задачи включена возможность резервного копирования и восстановления данных. В работе описана разработанная архитектура сервис-брокера, не зависящая от конкретной СУБД. На ее основе был разработан сервис-брокер (Раздел 3: Брокер) на языке Java, решающий поставленную задачу. Требуемые возможности резервного копирования и восстановления данных реализованы в виде специального Backup API (Раздел 4.1: Backup API), хранилищ данных (Раздел 5: Хранилища) и плагина командного интерфейса (Раздел 6: Плагин).

Список литературы

[1] Amazon. Amazon Simple Storage Service (S3) Documentation. — 2016. — URL: https://aws.amazon.com/documentation/s3/ (online; accessed: 01.05.2016).

[2] Apache. Maven – Welcome to Apache Maven. — 2016. — URL: https://maven.apache.org/ (online; accessed: 30.04.2016).

[3] BOSH. BOSH. — 2016. — URL: https://bosh.io/ (online; accessed: 01.05.2016).

[4] Cloud Foundry. 100-day Challenge 002: Running postgresql-cf-service-broker on Cloud Foundry | Cloud Foundry. — 2016. — URL: https://www.cloudfoundry.org/100-day-challenge-002-running-postgresql-cf-service-broker-cloud-foundry/ (online; accessed: 30.04.2016).

[5] Cloud Foundry. Cloud Foundry. The industry standard platform for cloud applications. — 2016. — URL: http://cloudfoundry.org/ (online; accessed: 29.04.2016).

[6] Cloud Foundry. Overview | Cloud Foundry Docs. — 2016. — URL: http://docs.cloudfoundry.org/services/overview.html (online; accessed: 01.05.2016).

[7] Cloud Foundry. Service Broker API v2.8 | Cloud Foundry Docs. — 2016. — URL: http://docs.cloudfoundry.org/services/api.html (online; accessed: 30.04.2016).

[8] Cloud Foundry. cf Command Line Interface (CLI) | Cloud Foundry Docs. — 2016. — URL: http://docs.cloudfoundry.org/cf-cli/ (online; accessed: 30.04.2016).

[9] ElephantSQL. ElephantSQL - PostgreSQL as a Service. — 2016. — URL: https://www.elephantsql.com/ (online; accessed: 30.04.2016).

[10] IBM. IBM Bluemix - Next-Generation Cloud App Development Platform. — 2016. — URL: https://console.ng.bluemix.net/ (online; accessed: 30.04.2016).

[11] Pivotal. Pivotal Cloud Foundry | Pivotal. — 2016. — URL: http://pivotal.io/platform/ (online; accessed: 30.04.2016).

[12] PostgreSQL. PostgreSQL: Documentation: 9.5: pg_restore. — 2016. — URL: http://www.postgresql.org/docs/9.5/static/app-pgrestore.html (online; accessed: 01.05.2016).

[13] SAP. Overview | SAP HANA Cloud Platform. — 2016. — URL: https://hcp.sap.com/index.html (online; accessed: 30.04.2016).

[14] Spring. Spring Framework. — 2016. — URL: https://projects.spring.io/spring-framework/ (online; accessed: 30.04.2016).